

Bericht TT2P 1

Steffen Brauer, André Harms,
Florian Johannßen, Jan-Christoph Meier,
Florian Ocker, Olaf Potratz,
Torben Woggan

10.06.2012

Inhaltsverzeichnis

1 Grundlagen	2
2 Dynamische Programmierung	3
2.1 Policy Evaluation	3
2.2 Policy Improvement	4
2.3 Policy Iteration	5
2.4 Value Iteration	5
3 Temporal Difference Learning	6
3.1 Gierige Wahl von Aktionen	7
3.2 On-Policy TD und Off-Policy TD	7
4 Fazit	8

Abbildungsverzeichnis

1 Pseudocode des Policy-Evaluation Algorithmus	4
2 Pseudocode des Value-Iteration Algorithmus	6

1 Grundlagen

Verstärkendes Lernen oder Bestärkendes Lernen (engl. Reinforcement Learning), ist der Überbegriff für eine Reihe von Methoden zum Maschinellen Lernen (engl. Machine Learning). Beim Verstärkenden Lernen gibt es keine Vorgabe von Trainingsbeispielen, der Agent kann nur aus den eigenen Erfahrungen lernen. Ein Agent befindet sich immer in einer Umwelt, über die er Informationen besitzen muss. Ein real existierender Agent (im Gegensatz zu einem Agenten, der nur in einer Simulation virtuell vorhanden ist) nimmt Informationen über die Umwelt mit Sensoren wahr. Auf die Informationen kann er mit Aktionen reagieren, z.B. ein mechanisches Teil bewegen. Das Handeln des Agenten besteht somit aus einer Folge von Aktionen. Das Ziel des Verstärkenden Lernens ist nun, dass der Agent selbständig erkennen kann, welche Aktion die günstigste ist. Dies muss er lernen können. Dieser Lernprozess basiert auf positiven Belohnungen und negativen Belohnungen (Kosten). Höhere Belohnungen werden vom Agenten bevorzugt. Der Agent lernt, wie die Aktionen belohnt werden, somit kann er in Zukunft bessere Entscheidungen treffen (Verstärkung). Die Verstärkung kann auch erst zu spät einsetzen (z.B. nach Ende eines Spiels), so dass das gelernte Wissen erst später eine Rolle spielt (z.B. beim nächsten Spiel).

Konkret wird vom Zeitpunkt t , dem Zustand s_t , der Aktion a_t und der Belohnung (Reward) r_t zum Zeitpunkt t gesprochen. Der Agent wählt zum Zeitpunkt t eine Aktion a_t und gelangt dadurch von Zustand s_t in Zustand s_{t+1} und erhält daraufhin eine Belohnung r_{t+1} . Im Zustand s_{t+1} wählt er nun wiederum eine Aktion a_{t+1} .

Die Belohnungen und der Folgezustand werden dabei als Teil des Modells der Umgebung angesehen. Allerdings ist die Umgebung im Allgemeinen nicht-deterministisch. Das Ausführen einer bestimmten Aktion in einem bestimmten Zustand muss somit nicht immer im gleichen Folgezustand resultieren. Der Übergang in die Folgezustände basiert aber immer auf den gleichen Wahrscheinlichkeiten, die sich im Laufe der Zeit auch nicht verändern, man sagt die Umgebung ist stationär.

Jeder Agent verfolgt eine bestimmte Strategie (engl. Policy), die definiert, welche Aktionen abhängig vom Zustand ausgeführt werden. Dabei unterscheidet man zwischen der stochastischen Strategie und der deterministischen Strategie. Die zuerst erwähnte Strategieart weist jeder Aktion a eine bestimmte Wahrscheinlichkeit zu, mit der sie ausgehend vom Zustand s ausgeführt wird. Die deterministische Strategie ordnet stattdessen jedem Zustand eine bestimmte Aktion zu. Das Ziel eines Agenten besteht darin, seine Belohnung über die Gesamtlaufzeit zu maximieren und damit die optimale Strategie π^* zu erlangen. Dabei hängt π^* von der Belohnungsfunktion ab, welche die Güte einer Entscheidungssequenz ermittelt. Bei episodischen Aufgaben endet jeder abschließender Schritt T mit der Berechnung der Belohnung R_t für die entsprechende Episode, die wie folgt ermittelt wird: $R_t = r_{t+1} + r_{t+2} + r_{t+3} \dots r_T$. Bei fortlaufenden Aufgaben hat man das Problem, dass die Belohnungen und Kosten ins unendliche steigen könnten. Um dies zu vermeiden, wird das sogenannte *discounting* verwendet. Dieses Prinzip schwächt zukünftige Belohnungen. Hierzu wird eine discounting-rate $0 \leq \gamma \leq 1$ ein-

geführt, welche dafür sorgt, dass die Gesamtbelohnung begrenzt wird und nicht ins unendliche steigt. Die nächste Formel zeigt die Verwendung der counting-rate für die Berechnung der Belohnung. $R_t = \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^2 r_{t+3} \dots r_T$

2 Dynamische Programmierung

Dynamische Programmierung ist eine Methode zum algorithmischen Lösen von Optimierungsproblemen und kann als Spezialfall von Reinforcement Learning angesehen werden. Dynamische Programmierung kann erfolgreich eingesetzt werden, wenn das Problem aus mehreren gleichartigen Teilproblemen besteht. Dabei muss sich eine optimale Lösung des Problems aus optimalen Lösungen der Teilprobleme zusammensetzen. Zuerst werden die optimalen Lösungen der kleinsten Teilprobleme berechnet. Diese werden dann geeignet zu einer Lösung eines nächstgrößeren Teilproblems zusammengesetzt. Diese Schritte werden wiederholt. Einmal berechnete Teilergebnisse werden gespeichert, um bei nachfolgenden Berechnungen gleichartiger Teilprobleme auf diese Zwischenlösungen zurückgreifen zu können, anstatt eine neue Berechnung anstellen zu müssen.

Die Voraussetzung, um dynamische Programmierung anwenden zu können ist ein vollständiges Modell. Dies bedeutet, dass alle Zustände inklusive ihrer Belohnungen und Folgezustände in Abhängigkeit von Ausgangszustand und einer Aktion bekannt sind. Somit sind $\mathcal{P}_{ss'}^a$ und $\mathcal{R}_{ss'}^a$ bekannt und müssen nicht erst durch Erfahrungen geschätzt werden.

2.1 Policy Evaluation

Policy Evaluation beschreibt die Berechnung von Zustandswerten bezüglich einer gegebenen Strategie (Policy) entsprechend der Bellman-Gleichung. Es werden für eine gegebene Strategie π die zugehörigen Wertefunktionen V^π berechnet. Dabei werden episodische Probleme betrachtet, so dass ein künstlicher Endzustand S_{final} und ein erweiterter Zustandsraum $S^+ = S \cup S_{final}$ (S mit S_{final}) definiert werden kann. Das Ergebnis wird iterativ ermittelt (bzw. approximiert). Hierzu wird eine aufeinander aufbauende Wertereihenfolge $V_0, V_1 \dots V_n$ für alle Zustände in $s \in S$ generiert. V_0 kann dabei beliebig für alle Zustände gewählt werden. Durch Iteration der Bellman-Gleichung erfolgt eine Näherung an die Wertfunktionen der Zustände:

$$V_{k+1}^\pi(s) = \sum_a \pi(s, a) \sum_{s'} \mathcal{P}_{ss'}^a [\mathcal{R}_{ss'}^a + \gamma V_k^\pi(s')] \quad (1)$$

Dies geschieht für alle Zustände in S^+ . Vor der Iteration wird V_k^π (beim Start ist $k = 0$) mit Null initialisiert. Die Iteration erfolgt von $k = 0$ bis unendlich, jedoch wird abge-

Generiert am: 7. Juni 2012
 Steffen Brauer, André Harms,
 Florian Johannßen, Jan-Christoph Meier,
 Florian Ocker, Olaf Potratz,
 Torben Woggan

brochen sobald die Abbruchbedingung erfüllt ist. Durch ausreichend langes Iterieren tastet man sich beliebig nahe an die echte Wertfunktion heran. Als Abbruchbedingung dient folgender Term:

$$\max_{s \in \mathcal{S}^+} |V_{k+1}(s) - V_k(s)| \quad (2)$$

Es wird hierbei der größte Differenz-Wert zwischen zwei Schritten genommen und auf Überschreiten des Schwellwertes überprüft.

In der Abbildung 1 ist der Pseudocode des Algorithmus aufgeführt.

```

Input  $\pi$ , the policy to be evaluated
Initialize  $V(s) = 0$ , for all  $s \in \mathcal{S}^+$ 
Repeat
   $\Delta \leftarrow 0$ 
  For each  $s \in \mathcal{S}$ :
     $v \leftarrow V(s)$ 
     $V(s) \leftarrow \sum_a \pi(s, a) \sum_{s'} \mathcal{P}_{ss'}^a [\mathcal{R}_{ss'}^a + \gamma V(s')]$ 
     $\Delta \leftarrow \max(\Delta, |v - V(s)|)$ 
until  $\Delta < \theta$  (a small positive number)
Output  $V \approx V^\pi$ 

```

Abbildung 1: Pseudocode des Policy-Evaluation Algorithmus

2.2 Policy Improvement

Grund für das Ermitteln einer Wertfunktion, ist das Bestreben eine bessere Strategie zu finden. Wenn V^π eine beliebige Wertfunktion für die Strategie π ist, ist es interessant, ob für einen Zustand s eine andere Aktion $a \neq \pi(s)$ gewählt werden soll, oder aber die alte Aktion $a = \pi(s)$ beibehalten werden soll. Es gibt verschiedene Möglichkeiten, diese Entscheidung zu treffen. Eine ist, eine beliebige Aktion a zu wählen und dann mit der bestehenden Strategie π fortzufahren. Der Wert durch Ausführen dieses Aktion a ist durch $Q^\pi(s, a)$ beschrieben. Die Frage, die sich stellt, ist: Ist $Q^\pi(s, a)$ besser als $V^\pi(s)$? Ist dies der Fall, sollte die ermittelte Aktion a der bestehenden Strategie vorgezogen werden.

Seien π und π' ein beliebiges Paar von Strategien, so dass für alle $s \in \mathcal{S}$ gilt:

$$Q^\pi(s, \pi'(s)) \geq V^\pi(s) \quad (3)$$

Dann ist Strategie π' mindestens so gut wie π , was wiederum bedeutet:

Generiert am: *Steffen Brauer, André Harms,*
 7. Juni 2012 *Florian Johannßen, Jan-Christoph Meier,*
Florian Ocker, Olaf Potratz,
Torben Woggan

$$V^{\pi'}(s) \geq V^{\pi}(s) \quad (4)$$

Mathematisch ausgedrückt kann die beste Aktion a für den Zustand s folgendermaßen gefunden werden:

$$\pi'(s) = \underset{a}{\operatorname{argmax}} \sum_{s'} \mathcal{P}_{ss'}^a [\mathcal{R}_{ss'}^a + \gamma V^{\pi}(s')] \quad (5)$$

Die Funktion *argmax* gibt den Funktionsparameter zurück, für den die Funktion den höchsten Wert ergibt*. In Gleichung 5 wird somit das a zurückgegeben, was den höchsten Wert ergibt und so eine Strategie-Verbesserung erreicht. π' stellt die verbesserte Strategie dar.

2.3 Policy Iteration

Falls sich die Strategie beim Strategie-Verbesserungsschritt (“Policy Improvement”) ändert muss diese neu ausgewertet werden. Somit müssen die V-Werte neu berechnet werden, da sie sich möglicherweise verändert haben.

Hieraus ergibt sich ein Verfahren mit dem schrittweise eine verbesserte Strategie gefunden werden kann. Es wird ausgehend von einer Strategie π_0 durch abwechselndes Ausführen des Evaluationsschrittes (E) und Verbesserungsschrittes (I) eine bessere Strategie π_1 berechnet. Im Evaluationsschritt werden die verbesserten V^{π_0} Werte berechnet, aus denen dann die Verbesserte Strategie im Verbesserungsschritt abgeleitet wird. Das Verfahren kann durch folgenden Ausdruck veranschaulicht werden:

$$\pi_0 \xrightarrow{E} V^{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} V^{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} V^{\pi_2} \xrightarrow{I} \dots \pi^* \quad (6)$$

Das Verfahren terminiert sofern sich die Strategie beim Verbesserungsschritt nicht mehr verändert.

2.4 Value Iteration

Eine wichtige Frage für die Laufzeit und Performanz des Verfahrens wie häufig bei der Strategie-Evaluation über die Bellman-Gleichung iteriert werden soll. Es ist durchaus denkbar, dass bereits nicht besonders exakte V-Werte zu einer Verbesserung der Strategie führen.

* Z.B. wenn $f(1) = 10, f(2) = 50, f(3) = 25$ dann würde $\underset{x}{\operatorname{argmax}} f(x)$ den Wert 2 zurückliefern

Der Ansatz der “Value Iteration” besteht darin nur eine Iteration der Bellman-Gleichung im Evaluationsschritt und direkt darauf folgend eine Strategie-Verbesserung durchzuführen. Die folgenden beiden Schritte werden abwechselnd durchgeführt (Gleichung 7 ist die Strategie Evaluation, Gleichung 8 ist die Strategie Verbesserung).

$$V_{k+1}^\pi(s) = \sum_a \pi_{k+1}(s, a) \sum_{s'} \mathcal{P}_{ss'}^a [\mathcal{R}_{ss'}^a + \gamma V_k^\pi(s')] \quad (7)$$

$$\pi_{k+1}(s) = \underset{a}{\operatorname{argmax}} \sum_{s'} \mathcal{P}_{ss'}^a [\mathcal{R}_{ss'}^a + \gamma V^\pi(s')] \quad (8)$$

Die beiden Gleichungen werden zu einer Gleichung zusammengefasst:

$$V_{k+1}(s) = \max_a \sum_{s'} \mathcal{P}_{ss'}^a [\mathcal{R}_{ss'}^a + \gamma V^\pi(s')] \quad (9)$$

Die Funktion *max* gibt den besten Wert zurück, der mit einem Parameter *a* erreicht werden kann*. Als Abbruchkriterien wird wieder wie bei der “Policy Evaluation” die Unterschreitung eines bestimmten Schwellenwertes festgelegt:

$$\max_{s \in \mathcal{S}^+} |V_{k+1}(s) - V_k(s)| \quad (10)$$

In der Abbildung 2 ist der Pseudocode des “Value Iteration”-Algorithmus aufgeführt.

```

Initialize  $V$  arbitrarily, e.g.,  $V(s) = 0$ , for all  $s \in \mathcal{S}^+$ 

Repeat
   $\Delta \leftarrow 0$ 
  For each  $s \in \mathcal{S}$ :
     $v \leftarrow V(s)$ 
     $V(s) \leftarrow \max_a \sum_{s'} \mathcal{P}_{ss'}^a [\mathcal{R}_{ss'}^a + \gamma V(s')]$ 
     $\Delta \leftarrow \max(\Delta, |v - V(s)|)$ 
until  $\Delta < \theta$  (a small positive number)

Output a deterministic policy,  $\pi$ , such that
 $\pi(s) = \arg \max_a \sum_{s'} \mathcal{P}_{ss'}^a [\mathcal{R}_{ss'}^a + \gamma V(s')]$ 

```

Abbildung 2: Pseudocode des Value-Iteration Algorithmus

3 Temporal Difference Learning

Temporal Difference Learning (TD-Learning) beschreibt eine Klasse von Verfahren, die ein Lernen allein aus Erfahrungen ermöglicht.

* Z.B. wenn $f(1) = 10, f(2) = 50, f(3) = 25$ dann würde \max_x den Wert 50 zurückgeben

Bei der Methode wird versucht die Zustands-Wertfunktion (V-Funktion) zu verbessern. Dazu wird davon ausgegangen, dass der Agent bereits eine vorläufige V-Funktion besitzt, welche aber noch nicht der Wirklichkeit entspricht. Weiterhin müssen Erfahrungen in der Bewertung von zurückliegenden Zuständen einfließen, da Erfahrungen womöglich erst nach einiger Zeit zu Tage treten. Somit ist davon auszugehen, dass die Wertfunktion in späteren Zuständen schon näher an der Wirklichkeit liegt. Da jedoch bis zur vollständigen Exploration der Umgebung die V-Funktion von der tatsächlichen Wertfunktion abweicht, wird diese geschätzt. Dies wird durch folgende Formel spezifiziert:

$$V^\pi = E_\pi r_{t+1} + \gamma V^\pi(s_{t+1}) \quad (11)$$

Die Zustands-Wertfunktion V für die Strategie π ist gleich dem Erwartungswert für den zu erwartenden Gewinn r_{t+1} und der über den Trace-Decay-Faktor verminderten V-Funktion des Folgezustands.

Der Trace-Decay-Faktor (γ) beschreibt, wie groß die Gewichtung von zukünftigen Entscheidungen ist. Dadurch werden nähere Zustände höher bewertet als spätere, bzw. zukünftige Belohnungen werden schwächer bewertet. Ist der Trace-Decay-Faktor gleich 1, so entspricht das TD-Learning der Monte-Carlo-Methode.

Weiterhin gibt es noch einen α -Wert, welcher angibt, wie nah der Folgewert an dem V-Wert des jetzigen Zustands annähert. Für hinreichend kleine α nähert sich die Methode dem korrektem V-Wert. Somit indiziert α die Lernrate zur Steuerung der Korrekturstärke, also in welchem Maße die Funktion verbessert oder geändert wird.

3.1 Gierige Wahl von Aktionen

Eine Strategie heißt gierig (greedy), wenn sie immer versucht die Aktion zu wählen, die zu einer optimalen Wertfunktion führt. Daraus resultiert das Problem, dass manche Aktionen nicht ausgeführt werden, obwohl die Erfahrung langfristig zeigen würde, dass sie eigentlich besser wären. Es ist natürlich nicht sinnvoll einen Agenten zufällig agieren zu lassen und ihn nicht auf seine Erfahrungen zurückgreifen zu lassen. Deshalb wird der Parameter ϵ eingeführt, welcher angibt, wie häufig nicht optimale Aktionen ausgewählt werden. Für $\epsilon = 0$ ist die Strategie gierig, sonst nennt man sie ϵ -gierig. Meist wird ein konstanter, kleiner ϵ -Wert gewählt.

3.2 On-Policy TD und Off-Policy TD

On-Policy und Off-Policy TD sind Strategieverbesserungsverfahren, die TD-Learning einsetzen(sic!).

Bei On-Policy TD(SARSA) wird der nächste Zustand anhand eines Pfades berechnet. Falls zufällig durch ϵ ein nicht optimaler Pfad gewählt wird, kann ein Zustand

möglicherweise komplett ausscheiden, obwohl er für die Zielerreichung essentiell wäre.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \quad (12)$$

Die Verbesserung der Q Funktion setzt sich hierbei aus dem alten Funktionswert plus einen durch α abgeschwächten Wert, der aus der zu erwartender Belohnung, sowie der Differenz zwischen Erwartungswert des nächsten und dieses Zustands.

Im Gegensatz dazu wird beim Off-Policy TD die Werteberechnung des nächsten Zustands nicht nur entsprechend eines Pfades durchgeführt, sondern jeweils immer der beste Nachfolgezustand einbezogen. Die Formel ändert sich hier nur im letzten Teil:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \quad (13)$$

4 Fazit